Pet clinic mandatory assignment report

Krisztian Szabo

November 13, 2019

`https://github.com/SzaboKrisztian/PetClinic`

Contents

1 Technologies employed

- **Java**: A high level object-oriented programming language that runs on the Java Virtual Machine.
- **Spring**: an open source Model-View-Controller (henceforth referred to as MVC) based application framework for Java, that is meant to facilitate the development of web applications. Spring boot is an opinionated approach that is mean to help the developer get going fast, by providing sensible defaults and including what are considered to be the best and most used third party libraries.

  MVC is a software design pattern, or as some call it, a high-level software architecture pattern, that is widely used in developing applications that feature a user interface. Its purpose is to decouple the data (the models) from the user interface (the views) and provide a means of communication between these two elements (the controllers). There are variations as to how the MVC pattern can be implemented, but in the more classic approach, the controller is the entry point to the program. This instantiates the required models and views, to which it passes a reference of itself, to facilitate communication from them back to the controller. The user only interacts with the views, which have the role of displaying themselves, and of accepting user input. This input is processed by the controller, which decides what action to take, like, for example, updating some values in the model.

  The MVC pattern achieves great decoupling between the different components, and allows for the models and views to be reused.
- **Thymeleaf**: A powerful and very flexible templating engine for Spring. This is used to programmatically generate the *.html* views and populate them with data.
- **Maven**: A build automation tool primarily used for Java projects. This allows for easy management of a project's dependencies.
- **Tomcat**: A pure Java HTTP web server which can run Java code. Maven uses this to launch and serve the web application.
- **Git**: The most popular version control tool to keep track of changes to a code base.
- **GitHub**: The most popular online hosting solution for Git repositories, which also allows easy collaboration among developers on a given code base.

2 Implementing the changes

I already had my development environment set up, with JDK, Maven, and git correctly installed and added to the environment variables. Cloning the repository to a local folder was also trivial.

The first real step in tackling this assignment was to identify where to make changes within the code base. By default, the Pet Clinic web application uses **hsqldb** as data storage. I found out that it works as an in-memory database, which creates its structure and populates it with data at runtime. So the place to modify the table structure is in the **src/main/resources/db/hsqldb/schema.sql** file and adding data to populate it in the **src/main/resources/db/hsqldb/data.sql** file.

The next thing to be modified was to add a new menu entry on the navigation bar for the future view that needed to be created. I found the navigation bar as a thymeleaf fragment in the **src/main/resources/templates/fragments/layout.html** file. I added an additional entry and pointed it to "/drugs" mapping, which for the moment does not yet exist.

Next task was to define the **Drug** class, which in our case should extend the existing **NamedEntity** class. It has to define the **petType**, **batchNumber**, and **expiryDate** attributes, and bind them to their respective SQL columns. The **id** and **name** attributes are taken care of in the **BaseEntity** and **NamedEntity** superclasses, respectively.

Then, I defined the **DrugRepository** interface, which defines some basic methods for retrieving and saving **Drug** entities from and to the database. Having done this, I moved to writing the **DrugController** class. In it, I added an instance of **DrugRepository** and defined the GET mapping for "/drugs". All this does is it inserts a list of all the available **Drug** entities into the model, and returns the **src/main/resources/templates/drugs/-drugs.html** view. Next I went ahead and actually created the view, which consists of a simple *html* table which is populated by thymeleaf by iterating over the list of **Drug** entities that was sent from the controller through the model.

After getting this to work, I realized that it would make logical sense to define a many-to-many relationship between the newly created drugs and the animal types, as one drug can most likely be used for several animals. To achieve this, I added a new table to the **schema.sql** file, that only serves the purpose of mapping this relationship, by the means of having only two columns, which are each a foreign key pointing to the **id** columns of the **drugs** and **types** tables, respectively. I then went ahead and modified the **Drugs** POJO to hold, instead of a single **PetType** attribute, a **Set<PetType>**. The last change needed to be made was in the view, for each drug displayed in the table, to iterate over all of its associated **PetType** objects and show them in a nicely formatted string.

Inspired by the way thymeleaf templates are used throughout the project to generate and populate web forms, I then went ahead and wrote **src/-main/resources/templates/drugs/addOrUpdateDrug.html** in a similar manner, to serve as the view for both adding and updating **Drug** entries. For this to work, I next had to define the four new mappings in the **Drug-Controller** class, a *GET* and a *POST* mapping for both the add and update functionality, and I added the relevant links in the main **drugs.html** view, namely an *"Add new drug"* button, and an *"Edit"* and a *"Delete"* link for each entry in the table. I also added a **DrugValidator** class to handle data validation in these two new views.

Lastly, after having talked about testing in class, I spent some more time understanding the integration tests that the original authors of this project have written. I then created a test suite called **DrugControllerTests.java** to test the new functionality that I've added to the project. This contains a setup method that is run before each test case, which populates the mocked repositories with some sample data, and then eight actual test cases:

- **testDrugList()**: tests the view drug list functionality
- **testInitCreationForm()**: tests the creation of the add new drug form
- **testProcessCreationFormSuccess()**: tests a correct submission of the add new drug form
- **testProcessCreationFormHasErrors()**: tests an invalid submission of the add new drug form
- **testInitUpdateForm()**: tests the creation of the update drug form
- **testProcessUpdateFormSuccess()**: tests a correct submission of the update drug form
- **testProcessUpdateFormHasErrors()**: tests an invalid submission of the update drug form
- **testDeleteDrugEntry()**: tests the delete drug functionality

3 Final thoughts

The process of making changes to this sample Spring project was quite enlightening, and I feel that I learned a lot from it. I got to see for the first time a code base that uses Spring's capabilities to the maximum, and I now realize that last semester's project could have been done in a much more efficient manner (though, to be fair, there were some limitations imposed on us by our teachers, for specific purposes). The most difficult part, so to speak, was understanding the code base. This took up the greatest part of my time spent on this project. Otherwise, after having understood what the original authors have built, implementing new functionality in the same style as theirs was fairly straight-forward.

I did hit a snag at one point, which took me a good few hours to debug. It had to do with getting Spring / Thymeleaf to correctly bind the multi-select box in the **addOrUpdateDrug.html** view to the **Drug** class' **petTypes** attribute, which is a collection of the type **Set<PetType>**. I didn't find any relevant documentation on this behavior, but apparently, when binding a web form to an object, Spring expects the POJO to have plain getters and setters for the attributes that are to be bound. I had originally written a slightly more complicated setter for this collection attribute in the **Drug** class, inspired by the authors' **Owner** class, and this lead to Spring not being able to insert the data submitted in the web form into the POJO's respective attribute.

Writing the integration tests was not completely easy either, mostly because of no previous experience with Spring's **MockMvc** testing tools. I did manage to do so for a series of test cases, but not for everything that I would have liked to test. For example, I didn't manage to test the multiple select box on the **addOrUpdateDrug.html** view. Even though I used Wireshark to see the exact parameters of the *HTTP POST* request that this web form input generates, and I reproduced them exactly within the test suite, Spring refused to convert from the submitted **String** parameters into the actual **Set<PetType>** needed. This conversion succeeds when running the app, but does not when running the tests. It might be that I'm making some wrong assumptions about the way **MockMvc** should be used in this case, but I couldn't find any information about this online.

4 Code listings

'schema.sql'

```
CREATE TABLE drugs (
  id              INTEGER IDENTITY PRIMARY KEY,
  name            VARCHAR(80),
  batch_number    VARCHAR(30),
  expiry_date     DATE
);
CREATE INDEX drugs_name ON drugs (name);

CREATE TABLE drugs_for_animal (
  drug_id         INTEGER NOT NULL,
  animal_type_id INTEGER NOT NULL
);
ALTER TABLE drugs_for_animal ADD CONSTRAINT
   fk_drugs_for_animal_types FOREIGN KEY (animal_type_id)
   REFERENCES types (id);
ALTER TABLE drugs_for_animal ADD CONSTRAINT
   fk_drugs_for_animal_drug FOREIGN KEY (drug_id) REFERENCES
   drugs (id);
```

'data.sql'

```
INSERT INTO drugs
VALUES (1, 'Drug one', 'B015L087', '2020-12-21');
INSERT INTO drugs
VALUES (2, 'Drug two', 'B005L027', '2022-06-29');
INSERT INTO drugs
VALUES (3, 'Drug three', 'B132L422', '2022-08-13');
INSERT INTO drugs
VALUES (4, 'Drug four', 'B073L132', '2021-10-10');
INSERT INTO drugs
VALUES (5, 'Drug five', 'B277L091', '2024-01-01');

INSERT INTO drugs_for_animal
VALUES (1, 1);
INSERT INTO drugs_for_animal
VALUES (1, 2);
INSERT INTO drugs_for_animal
VALUES (2, 3);
INSERT INTO drugs_for_animal
VALUES (2, 4);
INSERT INTO drugs_for_animal
VALUES (3, 5);
INSERT INTO drugs_for_animal
VALUES (4, 6);
INSERT INTO drugs_for_animal
```

```
VALUES (5, 1);
INSERT INTO drugs_for_animal
VALUES (5, 2);
INSERT INTO drugs_for_animal
VALUES (5, 5);
INSERT INTO drugs_for_animal
VALUES (5, 6);
```

'Drug.java'

```
@Entity
@Table(name = "drugs")
public class Drug extends NamedEntity {

  @ManyToMany(fetch = FetchType.EAGER)
  @JoinTable(name = "drugs_for_animal", joinColumns =
     @JoinColumn(name = "drug_id"), inverseJoinColumns =
     @JoinColumn(name = "animal_type_id"))
  public Set<PetType> petTypes;

  @Column(name = "batch_number")
  private String batchNumber;

  @Column(name = "expiry_date")
  @DateTimeFormat(pattern = "yyyy-MM-dd")
  private LocalDate expiryDate;

  public void setPetTypes(Set<PetType> petTypes) {
    this.petTypes = petTypes;
  }

  public Set<PetType> getPetTypes() {
    return this.petTypes;
  }

  public int getNrOfPetTypes() {
    return this.petTypes.size();
  }

  public String getBatchNumber() {
    return batchNumber;
  }

  public void setBatchNumber(String batchNumber) {
    this.batchNumber = batchNumber;
  }

  public LocalDate getExpiryDate() {
    return expiryDate;
  }

  public void setExpiryDate(LocalDate expiryDate) {
    this.expiryDate = expiryDate;
  }
}
```

'DrugRepository.java'

```
public interface DrugRepository extends Repository<Drug,
    Integer> {

  @Transactional(readOnly = true)
  Drug findById(Integer id);

  @Transactional(readOnly = true)
  List<Drug> findAll();

  Drug save(Drug drug);

  @Transactional
  void deleteDrugById(Integer id);
}
```

'DrugValidator.java'

```
public class DrugValidator implements Validator {

  private final String REQ = "required";

  @Override
  public boolean supports(Class<?> aClass) {
    return Drug.class.isAssignableFrom(aClass);
  }

  @Override
  public void validate(Object o, Errors errors) {
    Drug drug = (Drug) o;

    if (!StringUtils.hasLength(drug.getName())) {
      errors.rejectValue("name", REQ, REQ);
    }

    if (!StringUtils.hasLength(drug.getBatchNumber())) {
      errors.rejectValue("batchNumber", REQ, REQ);
    }

    if (drug.getExpiryDate() == null) {
      errors.rejectValue("expiryDate", REQ, REQ);
    }
  }
}
```

'DrugController.java'

```
@Controller
public class DrugController {

  private final String ADD_OR_UPDATE_DRUG_VIEW = "drugs/
      addOrUpdateDrug";

  private final DrugRepository drugRepo;
  private final PetRepository pets;

  public DrugController(DrugRepository drugs, PetRepository
      pets) {
    this.drugRepo = drugs;
    this.pets = pets;
  }

  @InitBinder("drug")
  public void initDrugBinder(WebDataBinder dataBinder) {
    dataBinder.setValidator(new DrugValidator());
  }

  @ModelAttribute("allPetTypes")
  public Set<PetType> populatePetTypes() {
    return new TreeSet<>(this.pets.findPetTypes());
  }

  @GetMapping("/drugs")
  public String seeDrugsList(Map<String, Object> model) {
    model.put("drugs", drugRepo.findAll());
    return "drugs/drugsList";
  }

  @GetMapping("/drugs/new")
  public String initAddDrugForm(Map<String, Object> model) {
    model.put("drug", new Drug());
    return ADD_OR_UPDATE_DRUG_VIEW;
  }

  @PostMapping("/drugs/new")
  public String processAddDrugForm(@Valid Drug drug,
      BindingResult result, Map<String, Object> model) {
    if (result.hasErrors()) {
      model.put("drug", drug);
      return ADD_OR_UPDATE_DRUG_VIEW;
    } else {
      this.drugRepo.save(drug);
      return "redirect:/drugs";
    }
  }

  @GetMapping("/drugs/edit/{drug_Id}")
  public String initUpdateDrugForm(@PathVariable("drug_Id")
      int drug_Id, Map<String, Object> model) {
    Drug drug = this.drugRepo.findById(drug_Id);
    model.put("drug", drug);
    return ADD_OR_UPDATE_DRUG_VIEW;
  }

  @PostMapping("/drugs/edit/{drug_Id}")
  public String processUpdateDrugForm(@Valid Drug drug,
      BindingResult result, ModelMap model) {
    if (result.hasErrors()) {
      model.put("drug", drug);
      return ADD_OR_UPDATE_DRUG_VIEW;
    } else {
      this.drugRepo.save(drug);
      return "redirect:/drugs";
    }
  }

  @GetMapping("/drugs/delete/{drug_Id}")
  public String deleteDrug(@PathVariable("drug_Id") int
      drugId) {
    this.drugRepo.deleteDrugById(drugId);
    return "redirect:/drugs";
  }
}
```

'drugsList.html'

```
<!DOCTYPE html>

<html xmlns:th="https://www.thymeleaf.org"
      th:replace="~{fragments/layout :: layout (~{::body},'
         drugs')}">

<body>

<h2>Drugs</h2>

<table id="drugs" class="table table-striped">
    <thead>
    <tr>
```

```
        <th style="width: 200px;">Name</th>
        <th>Pet type</th>
        <th>Batch number</th>
        <th style="width: 120px">Expiry date</th>
        <th></th>
        <th></th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="drug : ${drugs}">
        <td th:text="${drug.getName()}"></td>
        <td>
            <span th:if="${drug.getNrOfPetTypes()} > 0"
                  th:each="petType, iter : ${drug.getPetTypes
                     ()}"
                  th:text="${petType.getName()} + (!${iter.
                     last} ? ', ' : '')"></span>
            <span th:unless="${drug.getNrOfPetTypes()} > 0">
               none</span>
        </td>
        <td th:text="${drug.getBatchNumber()}"></td>
        <td th:text="${drug.getExpiryDate()}"></td>
        <td>
            <a th:href="'/drugs/edit/' + ${drug.getId()}">
               Edit</a>
        </td>
        <td>
            <a th:href="'/drugs/delete/' + ${drug.getId()}">
               Delete</a>
        </td>
    </tr>
    </tbody>
</table>

<a href="/drugs/new" class="btn btn-default">Add New Drug</a>

</body>
</html>
```

'addOrUpdateDrug.html'

```
<html xmlns:th="https://www.thymeleaf.org"
      th:replace="~{fragments/layout :: layout (~{::body},'
         drugs')}">

<body>

<h2>
    <th:block th:if="${drug.isNew()}">New </th:block>
    <th:block th:unless="${drug.isNew()}">Edit </th:block>
    Drug
</h2>

<form th:object="${drug}" class="form-horizontal" method="
   post">

    <div class="form-group has-feedback">
        <input th:replace="~{fragments/inputField :: input ('
           Name', 'name', 'text')}" />

        <input th:replace="~{fragments/inputField :: input ('
           Batch number', 'batchNumber', 'text')}" />

        <input th:replace="~{fragments/inputField :: input ('
           Expiry date', 'expiryDate', 'date')}" />

        <input th:replace="~{fragments/multiSelectField ::
           multiSelect ('Pet types', 'petTypes', ${
           allPetTypes})}" />
    </div>

    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <input type="hidden" name="id" th:value="*{id}" /
               >
            <button th:with="text=${drug.isNew()} ? 'Add drug
               ' : 'Update drug'"
                    class="btn btn-default" type="submit" th:
                       text="${text}"></button>
        </div>
    </div>
</form>

</body>
</html>
```

'DrugControllerTests.java'

```
@WebMvcTest(DrugController.class)
public class DrugControllerTests {

  @Autowired
  private MockMvc mockMvc;

  @MockBean
  private DrugRepository drugRepo;

  @MockBean
```

```java
private PetRepository petRepo;

@BeforeEach
void setup() {
  PetType dog = new PetType();
  dog.setId(1);
  dog.setName("dog");
  PetType cat = new PetType();
  cat.setId(2);
  cat.setName("cat");
  PetType rat = new PetType();
  rat.setId(3);
  rat.setName("rat");
  given(this.petRepo.findPetTypes()).willReturn(Lists.
     newArrayList(dog, cat, rat));

  Drug someDrug = new Drug();
  someDrug.setId(1);
  someDrug.setName("Some Drug");
  someDrug.setPetTypes(Sets.newLinkedHashSet(dog, cat));
  someDrug.setBatchNumber("SD1233B01");
  someDrug.setExpiryDate(LocalDate.now().plusYears(2));
  given(this.drugRepo.findAll()).willReturn(Lists.
     newArrayList(someDrug));
  given(this.drugRepo.findById(1)).willReturn(someDrug);
}

@Test
void testDrugList() throws Exception {
  mockMvc.perform(get("/drugs"))
      .andExpect(status().isOk())
      .andExpect(model().attributeExists("allPetTypes"))
      .andExpect(model().attributeExists("drugs"))
      .andExpect(view().name("drugs/drugsList"));
}

@Test
void testInitCreationForm() throws Exception {
  mockMvc.perform(get("/drugs/new"))
      .andExpect(status().isOk())
      .andExpect(model().attributeExists("allPetTypes"))
      .andExpect(model().attributeExists("drug"))
      .andExpect(view().name("drugs/addOrUpdateDrug"));
}

@Test
void testProcessCreationFormSuccess() throws Exception {
  mockMvc.perform(post("/drugs/new")
      .param("name", "New Drug")
```

```
        .param("batchNumber", "SBN4431L01")
        .param("expiryDate", "2025-12-12")
        .param("id", "")
  )
        .andExpect(status().is3xxRedirection());
}

@Test
void testProcessCreationFormHasErrors() throws Exception {
  mockMvc.perform(post("/drugs/new"))
        .andExpect(status().isOk())
        .andExpect(view().name("drugs/addOrUpdateDrug"))
        .andExpect(model().attributeHasErrors("drug"))
        .andExpect(model().attributeHasFieldErrors("drug", "
           name", "batchNumber", "expiryDate"));
}

@Test
void testInitUpdateForm() throws Exception {
  mockMvc.perform(get("/drugs/edit/{drug_id}", 1))
        .andExpect(status().isOk())
        .andExpect(model().attributeExists("allPetTypes"))
        .andExpect(model().attributeExists("drug"))
        .andExpect(model().attribute("drug", drugRepo.
           findById(1)))
        .andExpect(view().name("drugs/addOrUpdateDrug"));
}

@Test
void testProcessUpdateFormSuccess() throws Exception {
  mockMvc.perform(post("/drugs/edit/{drug_Id}", 1)
        .param("name", "Changed name")
        .param("batchNumber", "Makes no sense")
        .param("expiryDate", "2099-12-12")
        .param("id", "1")
  )
        .andExpect(status().is3xxRedirection());
}

@Test
void testProcessUpdateFormHasErrors() throws Exception {
  mockMvc.perform(post("/drugs/edit/{drug_Id}", 1))
        .andExpect(status().isOk())
        .andExpect(view().name("drugs/addOrUpdateDrug"))
        .andExpect(model().attributeHasErrors("drug"))
        .andExpect(model().attributeHasFieldErrors("drug", "
           name", "batchNumber", "expiryDate"));
}
```

```
  @Test
  void testDeleteDrugEntry() throws Exception {
    mockMvc.perform(get("/drugs/delete/{drug_Id}", 1))
        .andExpect(status().is3xxRedirection());
  }
}
```